ESD-TR-72-147, Vol. 3 MTR-2254, Vol. III

HARMONIOUS COOPERATION OF PROCESSES OPERATING ON A COMMON SET OF DATA, PART 3

by

L. J. LaPadula
D. E. Bell

DECEMBER 1972

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 671A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-71-C-0002

ESD-TR-72-147, Vol. 3 MTR-2254, Vol. III

HARMONIOUS COOPERATION OF PROCESSES OPERATING ON A COMMON SET OF DATA, PART 3

by

L. J. LaPadula
D. E. Bell

DECEMBER 1972

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 671A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-71-C-0002

FOREWORD

The work described in this report was carried out under the sponsorship of the Deputy for Command and Management Systems, Project 671A, by The MITRE Corporation, Bedford, Massachusetts, under Contract No. F19628-71-C-0002.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

[illegible]

MELVIN B. EMMONS, Colonel, USAF
Director, Information Systems Technology
Deputy for Command and Management Systems

ABSTRACT

A data-sharing scheduler is defined in terms of finite-state machine theory. Using the language and concepts of finite-state machines, we give precise definitions for the notions of "delayed," "blocked," "deadlock," "permanent blocking," and "sharing a datum"; these notions and their interrelationships lead to a characterization of a class of scheduler (unrestricted and nontrivial); a basic theorem of data sharing is stated and proved, and its implications are explored. We also give a narrative summary and discussion of the main result of the work suitable for the system designer or analyst.

ACKNOWLEDGMENT

This document is the third in a series of documents dealing with data sharing(6,1) produced under Project 671A, Multi-User Data Base Management.

The material presented herein, in Sections III through VI, will seem so abstract to the casual reader that he may well have difficulty imagining what may be the practical significance of the results obtained. We have attempted to guide the reader to a realization of the practical significance in the section entitled "Summary"; it is aimed at the reader who is a system designer or analyst.

By abstracting to the level of essential properties of interest and by the application of the simplest notions of finite-state machine theory, we have been able to obtain a very general and powerful result. With it we have been able both to evaluate the data-sharing models of Part 1 of this series(6) and to delineate guidelines for future investigation of data-sharing models.

We are greatly indebted to Mrs. Judith A. Clapp, MITRE Project Leader (671A), whose review of our initial draft led to a change in the definition of a trivial data-sharing scheduler; the resulting definition is more useful and more meaningful in its relation to real systems than the definition we had started with. Mrs. Clapp's review also gave us a number of insights which, we believe, made the final version more readable and more meaningful.

We are also indebted to Dr. Jonathan K. Millen, MITRE D73, who critically reviewed both the statement and proof of our Basic Theorem and suggested a number of interesting properties for future investigation.

TABLE OF CONTENTS

SECTION I

INTRODUCTION

Most data-sharing models which purport to avoid deadlock, including the models described in (6) (Part 1 of this series, which we shall hereinafter refer to as "HC1"), require an entering process to state which data elements it may use during its run. This requirement of prior knowledge seemed to be a logical candidate for weakening when we began attempting the formulation of a new model. A few initial failures in this direction suggested that perhaps prior knowledge of the requirements of a process is necessary to any data-sharing system which prevents deadlock. It is the intent of this paper to investigate the necessity of "prior knowledge" rigorously.

The sections of this paper are in the order which we felt would be most convenient for the majority of readers who may wish to apply the results reported herein to the practical problems of system design. We have, for example, presented the main result in narrative form together with some of its practical implications in the next section. The mathematical and developmental aspects of the work are then presented in subsequent sections.

The language of complete sequential machines was chosen to provide a rigorous framework for the investigation. Sections III through VI inclusive, assuming the material of the Appendix, develop concisely the relevant concepts involved in data sharing.

SECTION II

SUMMARY

PRÉCIS

We shall first, in narrative form, describe the main concepts which we have developed rigorously in the remainder of this paper. Then we shall state the main result of this paper. Finally, we shall briefly discuss the implications of the main result from the point of view of the system designer.

We consider the management of access to data in a system to be controlled by a Scheduler which is embedded in the system. The data in the data base are of two types with respect to the Scheduler: the data which may at some time in some mode of use be non-shareable simultaneously by two or more users (the critical data) and the data which are always shareable simultaneously by two or more users (the noncritical data). The function of the Scheduler is to avoid situations wherein critical data are being inappropriately shared.

Schedulers will be categorized broadly in two major ways in this paper. The first categorization involves the manner in which the data base is shared. A Scheduler is said to be nontrivial if it allows two or more users to have simultaneous access to critical data in the data base (perhaps two different data elements). On the other hand, a Scheduler is trivial if no more than one user at a time can have access to any elements in the critical data base. In a certain sense, a trivial Scheduler acts as a queue for the elements of the data base which cannot always be shared (i.e., as a queue for the critical data).

Two examples may illustrate what we mean to include in the designation "trivial Scheduler." If a Scheduler allows only one user access to the entire data base (one user at a time), no deadlock can occur and integrity of the data base can be preserved. In this case, the Scheduler acts as a queue of processes waiting for access to the entire data base: a request for access to a datum is granted if no other process has access to any element of the data base and is denied if any other process has access to some element of the data base. Suppose instead that a Scheduler always grants requests for access to data except for one datum, say d'; a request for access to d' is granted only when no other process has access to d'. Then, deadlock cannot occur since only requests for d' are ever denied; also, by assumption, simultaneous access to any element except d' will not compromise the integrity of the data base. In this case, the Scheduler acts as a queue with respect to d' and as a green light for the rest of the data base. This example is easily generalized so that D' contains a subset of the data base and only one process at a time may have access to elements in D'.

The second categorization of Schedulers concerns "prior knowledge" of the users' needs. In the models of HC1, for example, a process must state its claim list before it begins its run; for the remainder of the run, the system and that process share the knowledge of which data can be accessed and in what mode. This claim list is prior knowledge of the process's data needs during the run. Requests for data that are not properly claimed on the claim list are illegal and would be considered errors. In other models, an entering process can be required to supply information about its data needs and about the order in which the data will be used. In every case, the limiting values on a process's access capabilities, as a list of legal requests for data, would be considered prior knowledge. In this paper, we have used a more general concept which is explained below.

A Scheduler is said to be unrestricted if every request for access to any datum by any user is legal at any time. A Scheduler is restricted if some request for access to some datum by some user is illegal at some time. A restricted Scheduler has prior knowledge of the users' actions in that there is at every instant a list of acceptable (legal) requests that each user can make. The lists need not remain static as in the models of HC1; the only requirement is that there be such lists. In a real system, a user will know his options at each stage of his run and will thus be aware of the constraints placed on him. Notice, however, that unrestricted does not mean that every request for access will immediately be granted; indeed, we allow that an unrestricted Scheduler may deny a request for access to some datum by some user.

The forms that restricted Schedulers can take are many, as these examples will indicate. The models of HC1 are typical examples of restricted Schedulers. Another interesting example can be formulated in terms of the "name your elements in order" strategy for access. In this strategy all accessible elements are assumed to be ordered (i.e., there is a first, second, third, ...). A rule imposed on a process (user) is that it must always ask for access to elements in ascending order of their arrangement; it need not ask for elements which are contiguous in the ordering. The rule simply says that if a process is going to ask for an element b and a is the latest element it has acquired access to, then b must follow a in the ordering of the elements. If we assume that no elements may be shared and that the rule is enforced, then we have an example of a system which does not have a deadlock state and which does not share a datum; clearly, however, the Scheduler which manages this is restricted since a process having b may not legally ask for a, where a precedes b in the ordering of the elements.

Intuitively, an optimal Scheduler would be nontrivial (to effect as much "sharing" as possible) and unrestricted (to rid the users of the necessity of using some version of claims lists). However, the Basic Theorem stated and proved in Section V says essentially that

> "an unrestricted, nontrivial data-sharing Scheduler has a deadlock state or a state in which it shares a critical (unshareable) datum."

The major implications of the Basic Theorem are that an unrestricted, nontrivial, deadlock-free data-sharing system cannot guarantee integrity of the data base and that an unrestricted, nontrivial data-sharing system that does guarantee integrity of the data base cannot be deadlock-free. In other words, a design for a data-sharing system which must prevent deadlocks while guaranteeing integrity of the data base must involve either a restricted Scheduler or a trivial Scheduler.

CONCLUSIONS

We have arrived at two basic sets of conclusions, one having to do with data-sharing models and the other having to do with future investigations in this area.

The Basic Theorem of this paper indicates that in investigating strategies for achieving harmonious cooperation one must impose some restrictions on the users of the system. The particular form the restrictions take are of course important to the system designer since he will certainly wish to optimize one or more parameters and since different sets of restrictions could cause the data-sharing systems to appear quite different to the users.

Our conclusions about data-sharing models center around our evaluation of the generality of the models in HCl. The Basic Theorem further indicates that the models of HCl are essentially representative of the entire family of models which preserve integrity of the data base and which prevent deadlock and permanent blocking. The theorem indicates also that other significant models in the family are to be found by investigation of the conditions which make the models of HCl restricted schedulers--to wit, investigation of alternatives to static declaration by a process of its claim lists. Modifications to the strategies of HC1 or alternate strategies not having to do with this aspect of the model do not essentially change the model but may affect factors such as efficiency.

With regard to future work in this field, we believe that it will be fruitful to pursue the theoretical investigation begun in this paper--this belief seems justified in light of the significant result obtained from our simple use of basic finite-state machine theory. An obvious extension of this work would be the development of a characterization of a system with respect to permanent blocking, similar to the characterization presented herein with respect to deadlock.

SECTION III

DATA-SHARING SCHEDULERS

A number of data-sharing schemes (see (6, 4)) can be pictured as involving a black box for processing requests for data. Such a black box will typically receive as inputs

(1) requests for access to a datum in a particular mode (for example, write or read-only) and

(2) notifications that a particular datum has been returned to the data base.

The data-sharing scheme provides a method for the black box to process a request or a notification. The first capability is usually to determine whether the input signal is "legal." For example, if a process is forbidden by the scheme to request datum s in a write mode, a request from P for write access to datum s would normally cause the system to label the request an error. If an input is acceptable, it is processed. An acceptable request input produces either a granting response or a denying response. An acceptable datum-release input will generally cause internal bookkeeping; sometimes a release of a datum will cause the system to grant a request that had been denied earlier. For our purposes, another concept we must deal with is the idea that some record must be kept of the status of each datum and of each process: if this type of record is not kept, an actual data-sharing system could become hopelessly entangled.

The discussion in the preceding paragraph will be taken as justification for the definition of a data-sharing scheduler below. The material which follows uses the definitions found in the Appendix.

Let $\underline{P} = \{P_1, \ldots, P_n\}$ and $\underline{S} = \{s_1, \ldots, s_m\}$ be finite sets. An element of $\underline{P}$ will be called a "process" and an element of $\underline{S}$ will be called a "datum". A complete sequential machine $M = (K, \Sigma, \Delta, \lambda, \delta)$ is a <u>data-sharing scheduler</u> for $(\underline{P}, \underline{S})$ if

(1) $\Sigma \supseteq A \cup R$, where

  (a) A is a set whose elements are called "requests" and

  (b) $R = \{r(a): a \in A\}$ is a set whose elements are called "releases";

(2) $\Delta \supseteq \{\underline{0}, \underline{\text{error}}\} \cup \{G(a): a \in A\} \cup \{D(a): a \in A\}$;

(3) there is a unique state $q_0 \in K$ such that any $q \in K$ is of the form $q = \delta(q_0, J)$ for some tape J; and

(4) any state $q \in K$ specifies which processes have been granted access to which data and which processes have been denied access to some datum and have not yet been granted access in this state--that is, every state q records the current data allocation.

The elements of A are to be understood as requests for access to data. We shall denote by "A_{ij}" the set of all requests for datum s_j by process P_i. In other words, A is the disjoint union of all the sets A_{ij} where $1 \leqq i \leqq n$ and $1 \leqq j \leqq m$, and any element of A_{ij} will be understood to be a request by P_i for some (unspecified) kind of access to datum s_j. "a_{ij}" will be used to denote an element of A_{ij}. The elements of R are to be understood as notifications of the release of a datum: $r(a_{ij})$ is the notification that P_i relinquishes access to datum s_j as specified in request a_{ij}.

The output $G(a_{ij})$ is the granting of request a_{ij}; $D(a_{ij})$ is denying it.[1] The output error indicates an inappropriate input was supplied: for example, we would normally expect $\lambda(q_0, r(a_{ij})) = \underline{\text{error}}$ for every request a_{ij}. The output $\underline{0}$ is a go-ahead signal which is used when an input $r(a)$ yields neither an error output nor a $G(a_{ij})$ output.

With this precise definition of a data-sharing scheduler in hand, we are now in a position to begin a rigorous investigation of the major problems involved in data-sharing--blocking, deadlock, and integrity of the data base.

[1]The specification of a data-sharing scheduler does not take into account the action taken by the system when a request is denied. In particular, no assumption is made about whether a process is queued following a denial of its request.

SECTION IV

BLOCKING AND DELAYS

One of the problems that a data-sharing scheme must deal with is permanent blocking. Permanent blocking of a process P is a situation where it is possible for P to be "temporarily" denied access to some datum repeatedly for an indefinite period of time. Permanent blocking in a resource-sharing situation is discussed by Holt,(5) and examples of permanent blocking in a data-sharing situation are given in Section IV of HC1 (on pages 32 and 35). One solution to the problem of permanent blocking for data sharing is given in the second and third models of HC1. In essence, the solution's strategy is to close the system to entering processes when it first appears that some process P may be permanently blocked. The system then becomes internally identical to a system where no processes can enter. If each process terminates in a finite amount of time and if no processes can enter the system, then permanent blocking cannot occur by Theorem 11 of HC1. Hence the strategy used in HC1 guarantees that if a process P is given special attention to prevent it from becoming permanently blocked, then in some (unspecified) finite amount of time P will be able to proceed. The remainder of this section will provide both rigorous definitions of the concepts involved here and a brief analysis of their interrelations.

Let $M = (K, \Sigma, \Delta, \lambda, \delta)$ be a data-sharing scheduler for $(\underline{P}, \underline{S})$. For $q \in K$, we say that $\underline{P_i \text{ is waiting for access to } s_j \text{ at } q \text{ via } a_{ij}}$ (or briefly, "P_i is waiting for s_j at q") if there is a state q' and a tape J such that

(1) $\lambda(q', a_{ij}) = D(a_{ij})$;

(2) $\delta(q', a_{ij}J) = q$; and

(3) $\lambda(q', a_{ij}J)$ does not contain $G(a_{ij})$.

Heuristically, P_i is waiting for s_j at $q = \delta(q', a_{ij}J)$ if P_i was denied access to s_j at q' and access has not yet been granted to P_i.

The intuitive idea of permanent blocking is that a process P_i is waiting for s_j at q and that P_i could be waiting after any number of inputs (all "legal" in the sense that no one of them elicits an <u>error</u> output) has been processed. Our definition of permanent blocking will rely on the definitions below of "legal tapes" and of "k-blocked."

An input tape J will be called <u>legal</u> for $q \in K$ if $\underline{\text{error}} \notin \lambda(q, J)$. We say that $\underline{P_i \text{ is k-blocked at } q \text{ via } a_{ij}}$ if P_i is waiting for s_j at q via a_{ij} and there is an input tape J of length k which is legal for q such that $G(a_{ij}) \notin \lambda(q, J)$. That is, P_i is k-blocked at q via a_{ij} if it is possible for k "legal" inputs to be processed without granting the request a_{ij}. Clearly our definition of permanently blocked must imply k-blocked for every k.

We say that P_i is permanently blocked at q if P_i is waiting for s_j at q via some a_{ij} and P_i is k-blocked via a_{ij} for all $k > 0$. An equivalent statement of permanent blocking is given in Theorem 1.

Theorem 1: P_i is permanently blocked at q if and only if P_i is waiting for s_j at q via some a_{ij} and there is a sequence $\{I^k: 0 < k \text{ and } I^k \varepsilon \Sigma\}$ such that $G(a_{ij}) \not\subset \lambda(q, I^1 \ldots I^m)$ and $I^1 \ldots I^m$ is legal for q for every $m > 0$.

Proof: The condition is sufficient. To show P_i is k-blocked at q via a_{ij}, it suffices to consider the legal tape $I^1 \ldots I^k$.

Conversely, suppose P_i is permanently blocked at q via a_{ij}. For $s > 0$, let A_s denote the set of legal tapes J of length s such that $G(a_{ij}) \not\subset \lambda(q, J)$. Since P_i is permanently blocked via a_{ij}, each $A_s \neq \phi$ for $s > 0$. Also, if $s > 1$ and $J = J^*I \varepsilon A_s$ where $I \varepsilon \Sigma$, then J^* is a legal tape of length $s - 1$ such that $G(a_{ij}) \not\subset \lambda(q, J^*)$: $\lambda(q, J^*)$ is a subset of $\lambda(q, J)$. Hence every element J of A_s $(s > 1)$ is of the form $J = J^*I$ where $J^* \varepsilon A_{s-1}$ and $I \varepsilon \Sigma$. Since Σ is finite, A_{s+1} is finite provided A_s is finite. Moreover, A_1 is finite as a subset of Σ. Hence every A_s is finite. Define $F_s: A_s \rightarrow P(A_{s+1})$ (the "power set" of A_{s+1}--that is, the set of all subsets of A_{s+1}) by $F_s(J) = \{JI: JI$ is a legal tape for q and $G(a_{ij}) \notin \chi(q, JI)\}$.

The sets $\{A_s: 0 < s\}$ and $\{F_s: 0 < s\}$ have the property that for any $k > 0$ there is a sequence $\{J_1^{(k)}, \ldots, J_k^{(k)}\}$ such that $J_s^{(k)} \varepsilon A_s$ for $0 < s \leq k$ and $J_{s+1}^{(k)} \varepsilon F_s(J_s^{(k)})$ for $0 < s < k$. By the König Graph Theorem,(7) this property implies that there is a sequence $\{J_s: 0 < s\}$ such that $J_s \varepsilon A_s$ and $J_{s+1} \varepsilon F_s(J_s)$ for all $s > 0$. Set $J_1 = I^1$ and write $J_{s+1} = J_s I^{s+1}$ for all $s > 1$. The sequence $\{I^k: 0 < k\}$ of inputs establishes that the second condition of the theorem is necessary.

An interpretation of this theorem is that P_i is permanently blocked at q if it is possible for some sequence of legal inputs never to grant P_i access to an s_j for which P_i is waiting at q. That is to say, the theorem shows that the rigorous definition of permanent blocking given in this section conforms with the usual intuitive notion of permanent blocking.

As mentioned before, the strategy for avoiding permanent blocking in the models of HCl alters the rules of the game so that a process P in danger of permanent blockage is guaranteed that in some finite amount of time its denied request will be granted. Moreover, if one knew a maximum number of data transactions left in the run of each other running process, one could calculate the maximum length of "time" (measured in legal data transactions - a request or release) that P would have to wait before his request would be granted. This concept of the "maximum wait time" has some interesting relations with the various concepts of blocking.

We will say that <u>P_i is k-delayed at q via a_{ij}</u> if

(1) P_i is waiting for s_j at q via a_{ij};

(2) for all legal tapes J of length k, $G(a_{ij}) \varepsilon \lambda(q, J)$; and

(3) there is a legal tape J^* of length $k - 1$ such that $G(a_{ij}) \not\varepsilon \lambda(q, J^*)$.

Thus, P_i is k-delayed if it has been denied access to some s_j via some a_{ij} and if it is possible that it will have to wait for k legal inputs to be processed before it gains access but not possible that it will have to wait for more than k legal inputs to be processed. An immediate result is the following theorem.

<u>Theorem 2</u>: If P_i is k-delayed at q via a_{ij} for $k > 1$ and $I \varepsilon \Sigma$ such that $\lambda(q, I) \neq$ <u>error</u> or $G(a_{ij})$, then P_i is m-delayed at $\delta(q, I)$ via a_{ij} for some $1 \leqq m < k$.

<u>Proof</u>: P_i is still waiting for s_j at $\delta(q, I)$ via a_{ij}. Moreover, for any legal input tape J of length $k - 1$, IJ is a legal tape of length k so that $G(a_{ij}) \varepsilon \lambda(q, IJ) = \lambda(\delta(q, I), J)$. Suppose J^* is the longest legal tape such that $G(a_{ij}) \not\varepsilon \lambda(\delta(q, I), J^*)$.

Then the length of $J^* = n < k - 1$. Hence for any legal tape J of length $m = n + 1$, $G(a_{ij}) \varepsilon \lambda(\delta(q, I), J)$ by the maximality of the length of J^*. Furthermore, P_i is m-delayed at $\delta(q, I)$ via a_{ij} where $m = n + 1 \leqq k-1 < k$.

The precise relation between permanent blocking and k-delays is shown in the next theorem.

Theorem 3: P_i is permanently blocked at q via a_{ij} if and only if P_i is waiting for s_j at q via a_{ij} and P_i is not k-delayed via a_{ij} for any $k > 0$.

Proof: Suppose P_i is waiting for s_j at q and P_i is not k-delayed for any $k > 0$. Then for each $k > 0$, either

i) there is a legal tape J of length k such that $G(a_{ij}) \not\varepsilon \lambda(q, J)$, or

ii) for all legal tapes J^* of length $k - 1$, $G(a_{ij}) \varepsilon \lambda(q, J^*)$.

Let $k = 1$. Then ii) cannot hold since it must be true that $G(a_{ij}) \not\varepsilon \lambda(q, \phi)$. Thus i) holds for $k = 1$: there is a legal tape J of length 1 such that $G(a_{ij}) \not\varepsilon \lambda(q, J)$. Let $k = 2$. Then ii) cannot hold since we have just established the existence of a tape which contradicts ii) for $k = 2$. Proceeding inductively we find that for every $k > 0$ there is a legal tape J of length k such that $G(a_{ij}) \not\varepsilon \lambda(q, J)$. Therefore, P_i is k-blocked for every $k > 0$ and is therefore permanently blocked.

Conversely, assume that either P_i is not waiting for s_j at q via a_{ij} or P_i is k-delayed via a_{ij} for some $k > 0$. If P_i is not waiting for s_j at q via a_{ij}, then P_i is clearly not permanently blocked at q via a_{ij}. If P_i is k-delayed via a_{ij} for some $k > 0$, then for every legal tape J of length k $G(a_{ij}) \varepsilon \lambda(q, J)$ and therefore P_i is not permanently blocked since it is not k-blocked for this particular k.

The corollary below follows directly from Theorem 3.

<u>Corollary</u>: If P_i is k-delayed at q via a_{ij}, then P_i is not permanently blocked at q via a_{ij}.

Our last result involving delays and blocking is Theorem 4 below.

<u>Theorem 4</u>: If P_i is k-delayed at q via a_{ij} for some $k > 1$, then P_i is $(k - 1)$-blocked at q via a_{ij}.

<u>Proof</u>: That P_i is k-delayed implies that there is a legal tape J of length $k - 1$ such that $G(a_{ij}) \not\varepsilon \lambda(q, J)$, which by definition means that P_i is $(k - 1)$-blocked.

SECTION V

THE BASIC THEOREM

In this section we formally introduce the concepts of deadlock and of integrity of the data base. The section ends with the Basic Theorem which relates these concepts with the type of data-sharing scheduler necessary to avoid deadlock and to preserve the integrity of the data base.

We begin by making explicit the concept of "current access." For a state q, we say $\underline{P_i \text{ has access to } s_j \text{ at } q \text{ via } a_{ij}}$ if there is a state q', an input I, and a tape J with $r(a_{ij})$ not in J such that

(1) $\lambda(q', I) = G(a_{ij})$ and

(2) $\delta(q', IJ) = q$.

Heuristically, P_i has access to s_j at q if P_i was granted access at state q' and P_i has not relinquished access on some path from $\delta(q', I)$ to q. Note that we could have $I = a_{ij}$ or P_i might have made the request a_{ij} for s_j at some predecessor state of q'.

We are naturally concerned with the interaction of processes operating on a common set of data. If $\underline{S} = \{s_1, \ldots, s_m\}$ is a finite set of elements (intuitively, the data of the data base), we shall be interested in the nontrivial or critical portion of $\underline{S}$. We will define $\underline{C} \subseteq \underline{S}$ such that $\underline{C}$ includes only those elements of $\underline{S}$

which may at some time (that is, in some state) be considered non-shareable; $\underline{S} - \underline{C}$ then includes only those elements which may always be shared. Set $\underline{C} = \{s_j$: there is a state $q(j) \varepsilon K$ and an integer $i(j)$ such that

(1) $P_{i(j)}$ has access to s_j at $q(j)$, and

(2) $\lambda(q(J), a_{ij}) = D(a_{ij})$ for every $i \neq i(j)$ and every $a_{ij} \varepsilon A_{ij}\}$.

We must next define deadlock in the present context. Deadlock is defined here in precisely the expected manner. A state q is a <u>deadlock state</u> if there are sets $\{P_{i_1}, \ldots, P_{i_k}\}$ of processes and $\{s_{j_1}, \ldots, s_{j_k}\}$ of data such that each P_{i_t} has access to s_{j_t} at q and each P_{i_t} is waiting for $s_{j_{t+1}}$ at q (where $s_{j_{k+1}}$ is defined to be s_{j_1}).

Note that if q is a deadlock state then each P_{i_t} involved is permanently blocked at q. On the other hand, if P_i is permanently blocked at q via a_{ij}, then q is not necessarily a deadlock state (see HC1, page 35). Thus, a data-sharing scheduler which does not allow permanent blocking does not have a deadlock state, but a scheduler which has no deadlock state may still allow permanent blocking.

The object of this section is to characterize "unrestricted" data-sharing schedulers which do not have deadlock states. A conjectural characterization might be that data-sharing schedulers which have no prior knowledge of the running processes' needs and which do not have deadlock states either do not share the data base or they do not preserve the integrity of the data base, in the sense of HC1. This conjecture is in fact true, if the concepts involved are taken to be as we define them below.

To formalize the idea of "prior knowledge," we introduce the concept of an unrestricted[2] data-sharing scheduler. A data-sharing scheduler M is <u>unrestricted</u> if for every state q and every $a_{ij} \varepsilon A$, $\lambda(q, a_{ij})$ = <u>error</u> implies that P_i has access to s_j at q via some request $\hat{a}_{ij} \varepsilon A_{ij}$. Loosely speaking, M is unrestricted if no request by a process P is illegal except (possibly) a request for a datum to which P already has access.

We will call M a <u>trivial</u>[3] data-sharing scheduler if one or both of the following conditions hold:

(1) for every state q and every pair (P_{i_1}, s_{j_1}) and (P_{i_2}, s_{j_2}) of $\underline{P} \times \underline{C}$ where $i_1 \neq i_2$, P_{i_1} has access to s_{j_1} at q implies that P_{i_2} does not have access to s_{j_2} at q;

(2) $|\underline{C}|$[4] $\leqq 1$.

---

[2]This term is not to be confused with the term "nonrestricted sequential machine," (see, for example, (2)) a synonym for "complete sequential machine."

[3]Not to be confused with a trivial finite-state machine (2).

[4]$|X|$ means the cardinality of X; i.e., the number of elements of X.

M is nontrivial if it is not trivial; that is, M is nontrivial if and only if $|\underline{C}| > 1$ and there is a state q, processes P_{i_1} and P_{i_2} where $i_1 \neq i_2$, and data s_{j_1} and s_{j_2}, both in $\underline{C}$, such that P_{i_1} has access to s_{j_1} at q and P_{i_2} has access to s_{j_2} at q. Intuitively, a data-sharing scheduler is trivial if it "shares" data by allowing only one process at a time access to $\underline{C}$, the critical subset of the data set $\underline{S}$.

We say that M shares a datum if there is a state q, processes P_{i_1} and P_{i_2}, and a datum $s_j \in \underline{C}$ such that P_{i_1} and P_{i_2} have access to s_j at q. This formalization relates to integrity of the data base in the following way: in the case that M shares a datum s_j and s_j should not be shared (consider two processes simultaneously using s_j in write-mode as in HC1), then integrity of the data base will, in general, be destroyed. Thus, "sharing a datum" is a restricted converse of the concept of integrity of the data base.

We are now ready to state the conjectural characterization given above in rigorous terms and to prove its validity.

Basic Theorem: An unrestricted, nontrivial data-sharing scheduler M has a deadlock state or shares a datum.[5]

Proof: Suppose M does not share a datum. Pick a state q such that processes P_{i_1} and P_{i_2} have access to s_{j_1} and s_{j_2} (respectively) at q, where s_{j_1} and s_{j_2} are elements of $\underline{C}$: such a choice is possible since M is nontrivial. Furthermore, $j_1 \neq j_2$ since M does not share a datum. Let J

[5]Recall that a data-sharing scheduler has been defined to be a complete sequential machine.

be the tape $a_{i_1 j_2} \cdot a_{i_2 j_1}$. Clearly J is a legal tape, and $\lambda(q, J) = D(a_{i_1 j_2}) \cdot D(a_{i_2 j_1})$ since M does not share a datum. Now P_{i_1} is waiting for s_{j_2} and P_{i_2} is waiting for s_{j_1} at $\delta(q, J)$. In addition, P_{i_t} has access to s_{j_t} at $\delta(q, J)$, $t = 1, 2$. Thus, $\delta(q, J)$ is a deadlock state and the theorem is proved.

This theorem is the principal result of this paper. Its implications will be discussed in the next section.

SECTION VI

IMPLICATIONS OF THE THEOREM

The Basic Theorem shows the relation between deadlock and integrity of the data base in an unrestricted nontrivial scheduler: either deadlock or simultaneous access for some $s_j \varepsilon \underline{C}$ must be allowed. Hence, an unrestricted, nontrivial scheduler which protects the integrity of the data base has a deadlock state. On the other hand, an unrestricted, nontrivial, deadlock-free scheduler cannot guarantee integrity of the data base unless $\underline{C} = \phi$ (that is, unless there is no critical set of data). In case the system being considered uses the strategy of replication of a datum s_j whenever s_j must be shared, we consider this to be a case of "M shares a datum." In practice, such a strategy is costly and causes great difficulty (see, for example, the discussion by Gray(4)).

We assume for the remainder of this section that we are dealing with a system wherein $\underline{C} \neq \phi$.

If integrity of the data base and a guarantee against deadlock are necessary, then at least one of the conditions "unrestricted" and "nontrivial" must be weakened. Weakening "nontrivial" leaves a trivial scheduler, where at most one process at a time has access to the entire critical set of data. Weakening "unrestricted" means allowing for the possibility that some requests (besides those for data already accessed) are forbidden. In a practical situation, then, a user of a shared data base must have a set of guidelines to tell him what data he can attempt to access. This set of guidelines could vary dynamically or could remain static. If it remains static, then obvious alternatives are that either the user stated his own limits when he began his run (as in the model of HCl), or the system defined his limits for him, perhaps for that particular run only or because the user has associated with him some access set which is invariant. If the guidelines vary dynamically (i.e., during a run), the system could decide to vary them on its own initiative or the user could request an alteration of his guidelines using a new set of request signals.[6]

[6]The models of HCl do allow a user to alter his guidelines, but only by reducing his claim lists. Increasing the claim list is presently a nonexistent option in those models.

APPENDIX

A SUMMARY OF COMPLETE SEQUENTIAL MACHINES

A complete sequential machine[7] is a 5-tuple $M = (K, \Sigma, \Delta, \lambda, \delta)$ where

K is a nonempty finite set of "states,"

Σ is a nonempty finite set of "inputs,"

Δ is a nonempty finite set of "outputs,"

λ is a function from $K \times \Sigma$ into Δ, and

δ is a function from $K \times \Sigma$ into K.

The function δ is called the "next-state" function and λ the "output" function. Elements of K will be denoted q or q_i and elements of Σ will be denoted I or I^j. The machine is called "complete" because both λ and δ are defined for every pair (q, I) in $K \times \Sigma$.

A complete sequential machine M can be thought of as a processor: when in a given state q, an input I causes M to change its state to the state specified by the next-state function (that is, to $\delta(q, I)$) and to emit the output $\lambda(q, I)$. From the next state $\delta(q, I)$, another input will cause a similar action by the machine M.

A tape of length k, where $k \geq 0$, is a (possibly empty) sequence of inputs $I^1, \ldots, I^k$. We extend the definitions of λ and δ so

[7]The material in this section is found in (2, 3). A similar concept of machine is presented in HCl to model a process.

that if M is in state q, $\delta(q, I^1 \ldots I^k)$ is the state of M after the sequence of inputs $I^1, \ldots, I^k$ has been processed by M and $\lambda(q, I^1 \ldots I^k)$ is the sequence of outputs produced when M has processed the tape $I^1 \ldots I^k$. For example, if $\lambda(q, I^1) = 1$ and $\lambda(\delta(q, I^1), I^2) = 3$, then $\delta(q, I^1I^2) = \delta(\delta(q, I^1), I^2)$ and $\lambda(q, I^1I^2)$ is the sequence 1, 3.

REFERENCES

1. D. E. Bell, "Harmonious Cooperation of Processes Operating on a Common Set of Data, Part 2," MTR 2254, Vol. II, The MITRE Corporation, Bedford, Massachusetts, 9 March 1972. ESD-TR-72-147, Vol. 2.

2. A. Gill, Introduction to the Theory of Finite-State Machines, New York, McGraw-Hill Book Company, Inc., 1962.

3. S. Ginsburg, An Introduction to Mathematical Machine Theory, Reading, Massachusetts, Addison-Wesley Publishing Company, Inc., 1962.

4. J. Gray, "Locking" in the Record of the Project MAC Conference on Concurrent Systems and Parallel Computations, Association for Computing Machinery, 1970.

5. R. C. Holt, "Comments on Prevention of System Deadlocks," Communications of the Association for Computing Machinery, XIV, 1971, 36-38.

6. L. J. LaPadula, "Harmonious Cooperation of Processes Operating on a Common Set of Data, Part 1," MTR 2254, Vol. I, The MITRE Corporation, Bedford, Massachusetts, 1972. ESD-TR-72-147, Vol. 1.

7. B. L. Osofsky, "A Generalization of Quasi-Frobenius Rings," Journal of Algebra, 4, 1966, 373-387.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) The MITRE Corporation P.O. Box 208 Bedford, Mass.	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

HARMONIOUS COOPERATION OF PROCESSES OPERATING ON A COMMON SET OF DATA, PART 3

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

L. J. LaPadula
D. E. Bell

6. REPORT DATE DECEMBER 1972	7a. TOTAL NO. OF PAGES 32	7b. NO. OF REFS 7
8a. CONTRACT OR GRANT NO. F19(628)-71-C-0002	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-72-147, Vol. 3	
b. PROJECT NO. 671A		
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) MTR-2254, Vol. III	
d.		

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Deputy for Command & Management Systems Electronic Systems Division (AFSC) L. G. Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

A data-sharing scheduler is defined in terms of finite-state machine theory. Using the language and concepts of finite-state machines, we give precise definitions for the notions of "delayed," "blocked," "deadlock," "permanent blocking," and "sharing a datum"; these notions and their interrelationships lead to a characterization of a class of scheduler (unrestricted and nontrivial); a basic theorem of data sharing is stated and proved, and its implications are explored. We also give a narrative summary and discussion of the main result of the work suitable for the system designer or analyst.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
DATA SHARING						
DEADLOCK						
FINITE-STATE MACHINES						
PERMANENT BLOCKING						